Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

OPTIMAL ROBUST PATH PLANNING IN GENERAL ENVIRONMENTS

R. E. Gomory
T. C. Hu
A. B. Kahng
G. Robins

December 1991
CSD-910082

Optimal Robust Path Planning in General Environments

R. E. Gomory†, T. C. Hu‡, Andrew B. Kahng and Gabriel Robins

UCLA CS Dept., Los Angeles, CA 90024-1596
† The Sloan Foundation, New York, NY 10111
‡ UCSD CS&E Dept., La Jolla, CA 92093-0114

Abstract

We address robust path planning for a mobile agent in a general environment by finding minimum cost source-destination paths having prescribed widths. Our main result is a new approach which efficiently finds optimal solutions using a network flow formulation. This is the first combinatorial algorithm for robust path planning which yields optimal solutions in polynomial time, and represents a significant departure from the conventional exhaustive graph search techniques. Our method not only handles environments with solid polygonal obstacles but also generalizes to arbitrary cost maps which may arise in modeling incomplete or uncertain knowledge of the environment. Simple extensions allow us to address higher-dimensional problem instances and minimum-surface computations; the latter is a result of independent interest. We use an efficient implementation to exhibit optimal path-planning solutions for a variety of test problems. The paper concludes with open issues and directions for future work.

1 Introduction

To effect its goals, an autonomous mobile agent will execute a sequence of source-destination navigational tasks in the agent's configuration space. For each navigational task, the *motion planning problem* asks for a minimum-cost feasible path [4] [6] [17] [21]. The cost of a given solution may depend on many factors, including distance traveled, time or energy expended, and hazard probabilities encountered along the path. In practice, additional constraints might arise from a dynamic environment (e.g., moving obstacles), incomplete knowledge of the terrain and a concomitant need to reason under uncertainty, or such physical considerations as gravity, friction, and actuator kinematics. A large body of work has addressed the motion planning problem; for an excellent survey, the reader is referred to Latombe [17]. We begin our exposition by reviewing several major problem formulations and solution methods.

The *basic motion planning problem* involves a rigid-body mobile agent moving in an environment that is populated by static polygonal obstacles. In this simple paradigm, the agent's path is defined by a sequence of translations and rotations of the rigid body. Motion planning then entails computing a path from source to destination which avoids collisions between the agent and the obstacles, i.e., the path must lie entirely in the so-called free space. This basic formulation has been augmented in several ways. For example, multiple agents may be present: this yields a dynamic environment with moving obstacles, thus motivating the concept of *configuration space-time* [18]. Other common extensions incorporate robot kinematics to establish constraints on the solution space based on joint articulation or the physical characteristics of control structures. Holonomic and non-holonomic classes of constraints are discussed by, e.g., Canny [4]. Other researchers have considered solution paths with limits on curvature [19], or wherein certain motions are prohibited (e.g., moving in "reverse gear"). Two important extensions – (i) the need to incorporate uncertainty into a motion planning formulation, and (ii) the requirement of an error-tolerant solution – have strongly motivated our present work; these respectively yield the notion of a general cost function in a given terrain, along with the notion of a robust path solution. Section 2 gives a formal development of these two concepts.

The complexity of motion planning increases as more constraints are added to the basic formulation. However, any strategy which addresses the basic formulation will usually generalize to more complicated formulations (this is true for the new approach proposed in this paper). Consequently, we find that approaches in the literature fall naturally into only a few main categories.

The most straightforward motion planning algorithms are based on *graph search* approaches. Such methods use the vertices of a graph to represent feasible points in the agent's configuration space; edges in the graph represent optimum subpaths (also called *canonical* paths) between these configurations. Variations include retraction-based methods based on Voronoi decomposition [20], and approximate cell decomposition [18]. For simple problem formulations, e.g., a point robot and convex polygonal obstacles, computational geometry techniques afford reasonably efficient solutions, as surveyed in [21]. Obviously, the complexity of any variant is dependent on the number of paths computed and the ease of computing each opti-

mum subpath. With most formulations, exhaustive graph search methods such as depth-first branch-and-bound or A* constitute the only known optimal algorithms. Such enumerative approaches have exponential time complexity and are too slow to be practical [4]; efficient heuristics must therefore be employed.

A common heuristic for reducing search complexity is the hierarchical solution method based on 2^k-trees (esp. quadtrees and octtrees) or dissections into triangular or rectangular cells [5] [17]. Unfortunately, the hierarchical solution approach cannot guarantee optimal solutions [25]. Alternatively, physical analog methods might be employed; these commonly involve a *potential field* which forces the agent to move toward the goal configuration while avoiding contact with obstacles [3] [9] [16]. Path-finding may then be accomplished by variational methods (e.g., [22]), or by classical gradient or subgradient algorithms. Other work has proposed using analogs of graph search algorithms, notably the best-first and depth-first approaches. The fundamental problem with each of these methods is that the output can be a locally minimum solution that is not globally optimum. No efficient variants of the physical analog approach that can guarantee a global optimum solution appear possible [17].

The main contribution of this paper is a new algorithmic approach to *optimal* robust path-planning in environments with arbitrary cost maps. Essentially, we discard the usual mix of shortest-path algorithms and graph search techniques, and instead employ a more general combinatorial approach involving network flows [7]. The crucial observation is that a minimum-cost *path* which *connects* two locations s and t is equivalent to a minimum-cost *cut-set* which *separates* two other locations s' and t'. We obtain our new motion planning methodology by using efficient network flow algorithms to exploit this duality between connecting paths and separating sets.

Our approach is guaranteed to find *optimum* solutions to the minimum-cost path and minimum-cost robust path problem formulations which we define below. A primary attraction is that our algorithm runs in polynomial time; in fact, it can be implemented in $O(N^2 \cdot \log N)$ time where N is the number of nodes in a discrete mesh representation of the environment. This is a significant speedup over exponential-time exhaustive graph search, which was previously

the only provably optimal method. Experimental results confirm that we can efficiently find optimal robust paths where current combinatorial methods are prohibitively expensive, and where variational or gradient heuristics only return local optimum solutions. Our new approach also has a number of other desirable features:

1. the intrinsic planarity and geometry of robot motion planning yields a layered, bounded-degree network representation, resulting in added efficiency of our implementation;
2. via incremental flow techniques, the method is suitable for dynamic or on-line problems, as well as situations where early knowledge of partial solutions is helpful (i.e., requiring an *anytime* algorithm [8]); and
3. the method generalizes to provide a solution to the classical Plateau problem on minimal surfaces [23], a result which is of independent interest [13].

Given these advantages, we believe that our new approach provides a new, practical basis for certain classes of autonomous agent planning problems.

The remainder of this paper is organized as follows. In Section 2, we formally define the problem of robust motion planning in a terrain with arbitrary cost function. Section 2 also develops our solution of robust motion planning via network flows, and describes an efficient implementation. Section 3 gives experimental results showing optimal width-d paths in environments ranging from random cost maps to the more traditional class of maps with solid polygonal obstacles. We conclude in Section 4 with several directions for future research.

2 Robust Motion Planning by Network Flows

We begin this section by establishing notation and terminology. Our development focuses on the duality between connection and separation which motivates the network flow approach.

2.1 Problem Formulation

We say that a subset R of the plane is *closed* if it contains its own boundary. R is *path-connected* if any two points of R may be joined by a continuous path in R, and R is *bounded* if it is a subset of some circle with a fixed finite radius. Finally, R is *simple* if it is homeomorphic to a disk, that is to say, R contains no holes. We now formally define a *region* as follows:

Definition: A *region* R is a simple, closed, path-connected and bounded subset of the real plane.

The boundary of R partitions the plane into three mutually disjoint sets: the inside of R, the outside of R, and the boundary B itself. We consider path planning in R from source S to destination T, with S and T being disjoint connected *subsets* of the boundary B. A *path* is defined as follows:

Definition: Given a region R with a boundary B, a *path* between two disjoint connected subsets $S \subset B$ and $T \subset B$ is a non self-intersecting continuous curve $P \subseteq R$ which connects some point $s \in S$ to some point $t \in T$.

Clearly the path P partitions R into three mutually disjoint sets: (i) the set of points of R lying strictly on the left side of P, which we denote by R_l (we assume that P is oriented in the direction from s toward t); (ii) the set of points of R lying on the right side of P, denoted by R_r; and (iii) points of P itself. This is illustrated in Figure 1. It is possible for at most one of R_l and R_r to be empty, and this happens exactly when P contains a subset of B between S and T. More precisely, R_l (resp. R_r) is empty if $P \supseteq B_l$ (resp. $P \supseteq B_r$), where B_l and B_r respectively denote the subsets of the boundary B lying clockwise and counterclockwise between S and T, i.e., $B_l = (B \cap R_l) - (S \cup T)$ and $B_r = (B \cap R_r) - (S \cup T)$.

As noted above, the goal of our work is to optimally solve motion planning when two practical constraints are incorporated: an *arbitrary* cost function defined over the region, and a *robust* path requirement. The idea of a *general environment* is much more realistic than a formulation which uses only solid polygonal obstacles, and follows very naturally from the mobile agent having imperfect (probabilistic, fuzzy, or incomplete) knowledge of its surround-

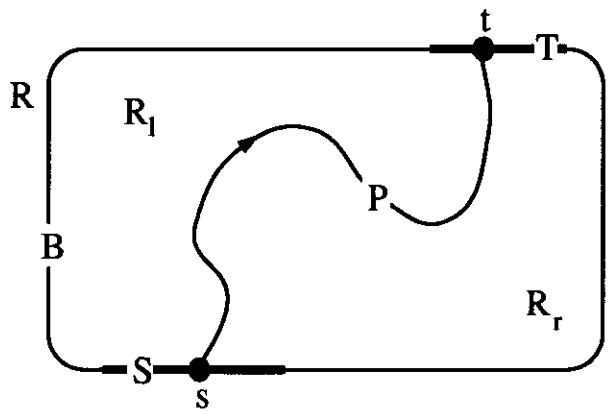

Figure 1: A path P between two points $s \in S$ and $t \in T$, where S and T are disjoint subsets of the boundary B of a region R.

ings. The imperfection of knowledge can be due to a variety of factors, including errors in positioning or orientation of the agent, staleness of information, and physical limitations of sensors. As an example, consider the following scenario: given probabilistic information about terrain difficulty (e.g., rivers, swamps, fields of landmines) and the locations of hostile entities, an agent wishes to travel with minimum probability of incurring a casualty or other damage. In this scenario, each point in the region will have an associated *weight*, or cost of traversal, corresponding to the level of danger. Multiple objectives may also be captured via this formulation: if the agent must reach his destination quickly, then the weight function will take into account both danger level and the time required to travel through each point. Note that this formulation subsumes the 0-1 cost function of a basic environment with solid polygonal obstacles and free space. Formally, given a region R, we define a weight function $w : R \rightarrow \Re^+$ such that each point $s \in R$ has a corresponding positive weight w_s.

In a general environment, i.e., an arbitrary region R with arbitrary weight function w, the *cost* of a path $P \subseteq R$ is defined to be the integral of w over P. Optimal motion planning entails minimizing this path integral. To find a minimum-cost path $P \subseteq R$ between two points on the boundary of R, one might be tempted to suppose that Dijkstra's shortest path algorithm [7] would provide a natural solution. However, a little reflection shows that application of Dijkstra's algorithm relies on an implicit assumption that the solution is an *ideal* path, i.e., a

path of *zero* width. The relevance of this caveat becomes clear as we now consider our second extension to the basic formulation, which is the requirement of a *robust* motion planning solution.

Due to errors in location, orientation and motion, a robot agent is never able to exactly follow a given prescribed path. A path is error-tolerant if the agent will not come to harm when it makes a small deviation from the path; in the above scenario, a path which requires the agent to tiptoe precisely between landmines is not as error-tolerant as a path which avoids minefields altogether. With this in mind, we say that a *robust* path is one which that is error-tolerant in the sense of maintaining a prescribed minimum width. This minimum-width formulation also addresses the fact that robot agents are physical entities with physical dimensions: because physical agents do not occupy point locations in the environment, the minimum-cost path of zero width that is computed by a shortest-path algorithm may be infeasible (e.g., while an insect might easily exit a room through a barred window, a person cannot easily do so). In general, the optimum path for an agent of width d_1 cannot be obtained by simply widening or narrowing the optimum path for an agent of width d_2.

We now establish the relationship between a minimum-width path requirement and the concept of *d-separation*. In the following, we use $ball(x, d)$ to denote the closed ball of diameter d centered at x, i.e., the set of all points at distance $\frac{d}{2}$ or less from x.

Definition: Given two disjoint subsets S and T of the boundary of a region R, a set of points $\hat{P} \subseteq R$ is a *width-d path* between S and T if there exist $s \in S$, $t \in T$ and a path P connecting s to t such that $\hat{P} \supseteq \bigcup_{x \in P} \{ball(x, d) \cap R\}$, i.e., $\hat{P}$ contains the intersection of R with any disk of *diameter d* centered about a point of P.

Just as the path P between S and T will partition R into R_l, R_r, and P, we say that the width-d path $\hat{P} \subseteq R$ between S and T also partitions R into three sets: (i) the set of points $\bar{R}_l = ((R - \hat{P}) \cap R_l) \cup B_l$, that is, the union of the left boundary B_l and all points in R that are to the left of $\hat{P}$; (ii) the set of points $\bar{R}_r = ((R - \hat{P}) \cap R_r) \cup B_r$; and (iii) the pointset $\hat{P}$ itself. We now obtain the definition of a d-separating path (see Figure 2):

Definition: Given two disjoint subsets S and T of the boundary of a region R, a set of points

$\bar{P} \subseteq R$ is a *d-separating path* between S and T if $\bar{P}$ is a width-d path such that any point of $\bar{R}_l$ is distance d or more away from any point of $\bar{R}_r$.

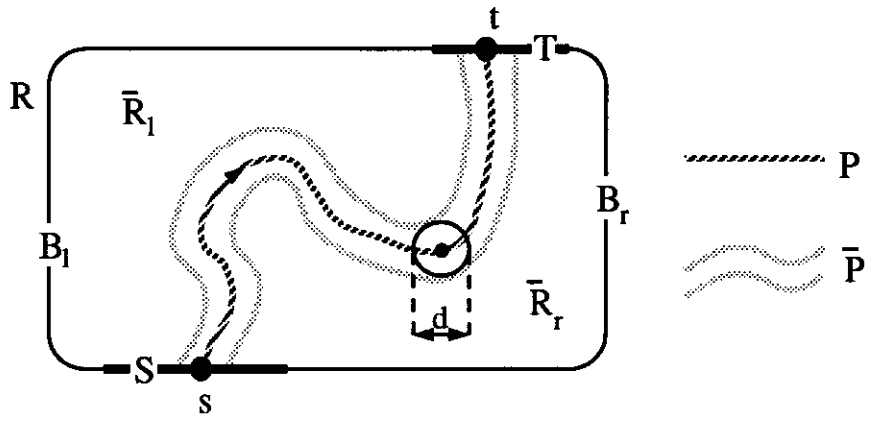

Figure 2: A d-separating path $\bar{P}$ of width d between two points $s \in S$ and $t \in T$ of the boundary of a region R. Here $\bar{R}_l$ is separated from $\bar{R}_r$ by a distance of d.

A d-separating path $\bar{P}$ between S and T is a *minimal d-separating path* between S and T if no subset of $\bar{P}$ satisfies the preceding definition. Because all points in R have positive cost and because we are interested in minimum-cost paths, the following discussion refers only to minimal d-separating paths. Given a specific width d, the d-robust motion planning problem can now be stated as follows:

d-Robust Motion Planning Problem (RMPP): Given an arbitrary region R with boundary B, weight function $w : R \rightarrow \Re^+$, a source $S \subseteq B$, a destination $T \subseteq B$, and a width d, find a d-separating path $\bar{P} \subseteq R$ between S and T which has minimum total weight.

Note that while our formulation specifies an arbitrary weight function that is integrated to yield path cost, in most practical situations the region is discretized relative to a given fixed grid or sampling granularity (see, e.g., [3]). Thus, in the present work, we will assume a fixed-grid representation of the environment. With this discrete RMPP formulation, the cost of a path is naturally defined to be the sum of the weights of the nodes contained in the path. We therefore have the following problem formulation:

d-Robust Motion Planning Problem in a Grid (RMPPG): Given a node-weighted

gridgraph $G = (V, E)$ with boundary $B \subset V$, a source $S \subseteq B$, a destination $T \subseteq B$, and a width d, find a d-separating path $\bar{P} \subseteq R$ which connects S and T and has minimum total weight.

Intuitively, as the granularity quantum approaches zero the solution for the RMPPG instance will converge to the solution for the corresponding continuous RMPP instance.

At this point, we observe that although RMPPG is a very natural problem formulation, it cannot be efficiently solved by traditional methods. Computational geometry techniques are suitable for regions with solid polygonal obstacles, but are ineffective given arbitrary cost maps. Hierarchical cell decomposition techniques cannot be used, since loss of information and suboptimal solutions will result. Finally, efficient graph algorithms such as Dijkstra's shortest-path algorithm also fail: when the required width d is greater than zero, the optimal path may self-intersect, and when the shortest-path algorithm attempts to increment a path, it cannot determine how much of the increment should be added to the path cost (see Figure 3).[1] Consequently, only exhaustive graph search techniques can be used. The ideas related to d-separation were first formalized and developed in [14].

We now use a network flow approach to develop an efficient, optimal algorithm for the RMPPG problem.

2.2 A Network Flow Based Approach

To solve the robust motion planning problem in a grid, we use ideas from the formulation of network flows in continua [15]. For completeness of the exposition, we briefly review several key concepts from network flows [10]. A *flow network* $G = (V, E)$ is a directed graph with no self-loops in which each edge $e \in E$ has a positive *capacity* $c(e) > 0$; edges that are not present in the network implicitly have capacity zero. There are two distinguished nodes in G, a *source*

[1] In an n-node edge-weighted graph $G = (V, E)$ with identified source $v_0 \in V$, the k^{th} phase of Dijkstra's algorithm, $k = 1, \ldots, n$, finds another node v_k for which the shortest pathlength d_{0k} in G is known; we know the optimum s-t pathlength when $v_k = t$. Although the RMPPG formulation above assumes a node-weighted G, we may easily obtain an edge-weighted graph (for all $v \in V$, add $\frac{w(v)}{2}$ to the weight of each edge incident to v) to which we may apply Dijkstra's algorithm. However, this transformation is correct only for computing the optimal zero-width path: Dijkstra's algorithm relies on the fact that d_{ij} can never be strictly less than $min_k(d_{ik} + d_{kj})$, but this fails to hold when paths have non-zero width, as shown in Figure 3. [15]

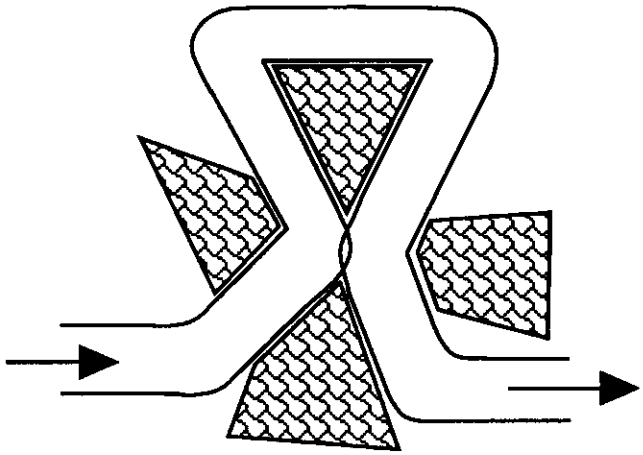

Figure 3: An optimal path with non-zero width may intersect itself; efficient shortest-path algorithms are therefore inapplicable to robust path planning.

$s \in V$ and a *sink* $t \in V$. A *flow* in G is a real-valued function $f : E \rightarrow \Re^+$ that satisfies the following properties:

1. **Capacity constraint:** For all $e \in E$, we require $f(e) \leq c(e)$ (when $f(e) = c(e)$ the edge e is called *saturated*).
2. **Flow conservation:** For all $v \in V - \{s, t\}$, we require $\sum_u f((u, v)) = \sum_w f((v, w))$.

The *value* of a flow f is given by $|f| = \sum_{v \in V} f((s, v))$. For a given network, the network flow problem is to find a flow of maximum value. An *s-t cut* in the network is a subset of edges whose removal disconnects s from t; the cost of the cut is obtained by summing the edge capacities in the cut. A well-known result from linear programming duality is the *max-flow min-cut* theorem [10]:

Theorem 1: Given a network G with identified source s and sink t, the value of the maximum s-t flow is equal to minimum sum of edge capacities of any s-t cut. Moreover, the edges of the minimum s-t cut are a subset of the saturated arcs in the maximum s-t flow. □

Recall our earlier observation that any s-t path will separate, i.e., cut, R_l from R_r. In particular, an inexpensive s-t path will correspond to an inexpensive cut between two appro-

priately chosen nodes s' and t'. Since a subset of the saturated edges in the maximum s'-t' flow will yield this s'-t' cut, it is natural for us to derive the desired s-t path via a maximum-flow computation in a network where edge weights correspond to travel costs. The remainder of this section describes how to accomplish this.

To transform robust motion planning in a region R into network flow, we first superimpose a mesh network topology over R, then assign node weights in this network according to the weighting function $w : R \rightarrow \Re^+$. This yields a representation that is essentially equivalent to the underlying RMPPG instance.

In order to guarantee a *robust* path solution, we need to ensure that any separating node set in the mesh topology will satisfy the width-d requirement. To this end, define the d-neighborhood of a node v in the mesh to be the set of all nodes that are at distance of d or less units away from v. We then modify the topology of our mesh network by uniformly connecting each node to all other nodes in its d-neighborhood, where d is the prescribed path width. The resulting network is called a *d-connected* mesh, and has the property that no nodeset of width less than d is a d-separating set. An illustration of this construction for $d = 2$ is given in Figure 4. We note that the concept of a d-neighborhood was first investigated by Gomory and Hu and reported in [15].

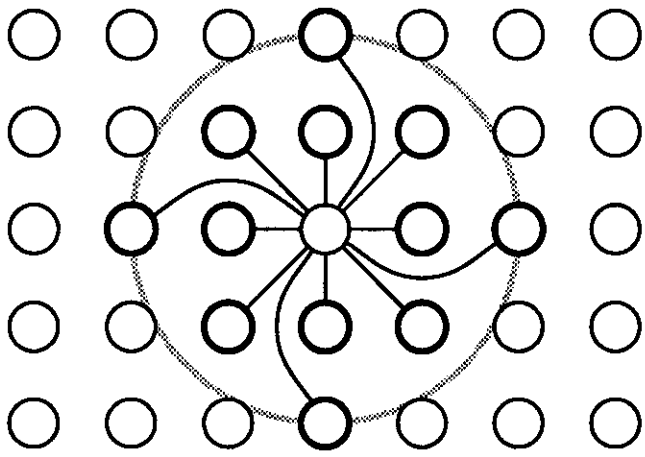

Figure 4: A node and its d-neighborhood; here d=2.

Finally, we choose the nodes s' and t' such that the s'-t' cut is forced to lie along some path between s and t. We accomplish this by making s' and t' respectively into a "super-source" and a "super-sink", then connecting each to a contiguous set of nodes corresponding to part of the boundary of the original region R. This completes the outline of our transformation; Figure 5 gives a high-level illustration of the construction.

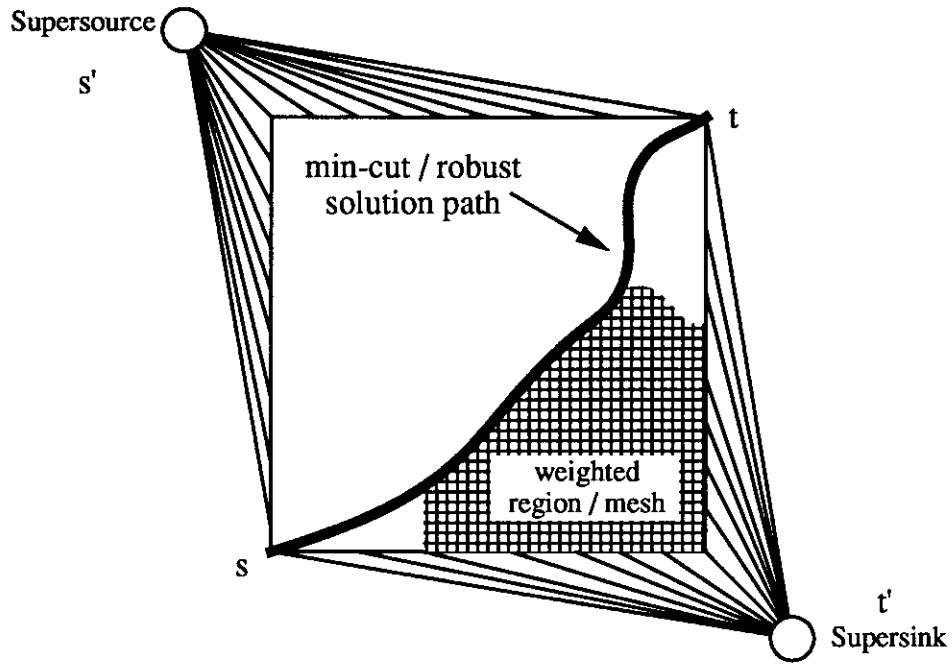

Figure 5: A robust motion planning instance transformed into a network flow instance.

Observe that up to this point, we have converted a motion planning instance to an undirected, *node-weighted* flow instance. However, the flow formulation above applies only to *edge-weighted* networks, i.e., networks with edge capacities. Therefore, we require a final step which will transform the node-weighted graph obtained thus far into an edge-weighted maximum flow instance. To accomplish this, we split each node $v \in V$ with weight w_v into two unweighted nodes v' and v'', then introduce a directed edge from v' to v'' with capacity w_v. Finally, we transform each edge $(u, v) \in E$ in the original graph into the two directed edges (u'', v') and (v'', u'), each with infinite capacity. Note that each edge (v', v'') of the resulting directed network will, when saturated, contribute the original node weight w_v to the value of

the minimum cut. This transformation is illustrated in Figure 6.

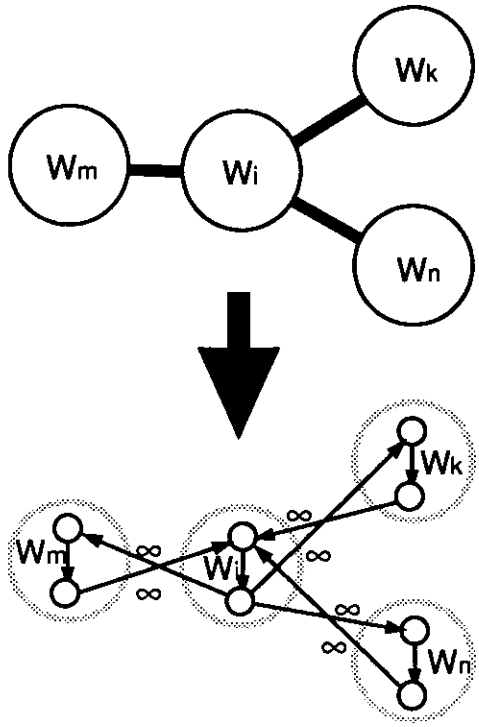

Figure 6: Transformation of node-weighted network to edge-weighted flow network.

The overall size of the network increases by only a constant factor via this last transformation: the final directed network will have only $2|V|$ nodes and $V + 2|E|$ edges. This implies that computing maximum flow on the transformed network will be asymptotically as fast as operating on the original network. A formal summary of our algorithm is given in Figure 7.

We conclude this section with the observation that the max-flow min-cut theorem [10] and the existence of efficient algorithms for maximum flow (e.g., [10] [7]) together imply the following:

Theorem 2: The method of Figure 7 outputs an optimal solution to the RMPPG problem in time polynomial in size of the mesh representation of the region R. □

Finding a d-robust path in a weighted region
Input: Region R, weight function $w : R \rightarrow \Re^+$, width d, grid size g, source s and destination t on boundary of R
Output: A d-separating path $\bar{P} \subseteq R$ connecting s and t
Create a d-connected mesh graph G of size $g \times g$ over R
Assign node weights in G according to weight function w
Set all boundary node weights to ∞
Transform node-weighted network G into edge-weighted network G'
Add super-source s' and super-sink t' to G'
Connect s' to B_l, the boundary nodes of R, clockwise from s to t
Connect t' to B_r, the boundary nodes of R, clockwise from t to s
Set capacities of all edges adjacent to s' or t' to ∞
Compute maximum s'-t' flow in G'
Output all nodes incident to edges in the minimum s'-t' cut of G'

Figure 7: Finding a d-separating path of minimum cost in an arbitrary weighted region, i.e., an optimal solution to the RMPPG problem.

2.3 A Practical Implementation

There are numerous algorithms for computing maximum network flow [10] [1] [15]. To demonstrate the viability of our approach, we have simply applied an existing implementation of Dinic's network flow algorithm [12]. Starting with an empty flow, the Dinic algorithm iteratively augments the flow in stages; the optimal flow solution is achieved when no flow augmentation is possible. Each stage starts with the existing flow, and attempts to "push" as much flow as possible along shortest paths from the source to the sink in an augmented network wherein each edge has capacity equal to the difference between its original capacity and its current flow value. After the current flow has been thus augmented, newly saturated edges are removed and the process iterates. Since there can be at most $|V| - 1$ such stages, each requiring time at most $O(|E| \cdot |V|)$, the total time complexity of the Dinic algorithm is $O(|E| \cdot |V|^2)$.

If we have a total of N nodes in our mesh graph, the time complexity of the Dinic algorithm is $O(N^3)$. In practice, more efficient flow algorithms are available [1]. For example, by using the network flow algorithm of [11], we obtain the following:

Theorem 3: For a given prescribed path width d, our method solves the RMPPG problem in

$O(N^2 \cdot \log N)$ time, where N is the number of nodes in the mesh representation of the region R.

Proof: Each node in the mesh induced by our method has no more than d^2 adjacent edges. This implies that $|E| = O(d^2 \cdot |V|)$. The network flow algorithm of [11] operates within time $O(|E| \cdot |V| \cdot \log(\frac{|V|^2}{|E|}))$. Setting $N = |V|$ and assuming that d is a constant, the overall time complexity of our method is therefore $O(N^2 \cdot \log N)$. □

The overall time complexity may be further reduced in cases where the terrain cost function may only take on values from a fixed bounded range. In this case, we may apply the maximum flow algorithm of [2] to obtain an overall time complexity of $O(N^2)$ for our method.

3 Experimental Results

Our current implementation integrates ANSI C code to transform an arbitrary robust motion planning instance into a maximum-flow instance; we then use the Fortran-77 Dinic code of [12] to compute the flow, and then invoke Mathematica [24] to draw the resulting path. We have tested our implementation on three classes of motion planning instances: uniformly weighted regions, environments with polygonal obstacles, and smooth random terrains. For each of these input classes, the boundary of the region is a rectangle, and we look for a path connecting s and t which are respectively in the top left and bottom right corners of the region.

A uniformly weighted region has all node weights equal to the same constant. In such an instance we expect the solution path to resemble a straight line between s and t, with the straightness of the line improving as the mesh resolution and the width d both increase. Experimental results confirmed this behavior.

Our test environments with polygonal obstacles are populated by random polygons of random sizes, placed in random locations throughout the region. Nodes in the clear areas are uniformly assigned zero weight, while nodes corresponding to subregions occupied by obstacles have infinite weight. In such an environment, changing the robust path parameter d may dramatically affect the optimum path topology with respect to the obstacles. This phenomenon

was indeed confirmed by our experiments, as illustrated in Figures 8 and 9.

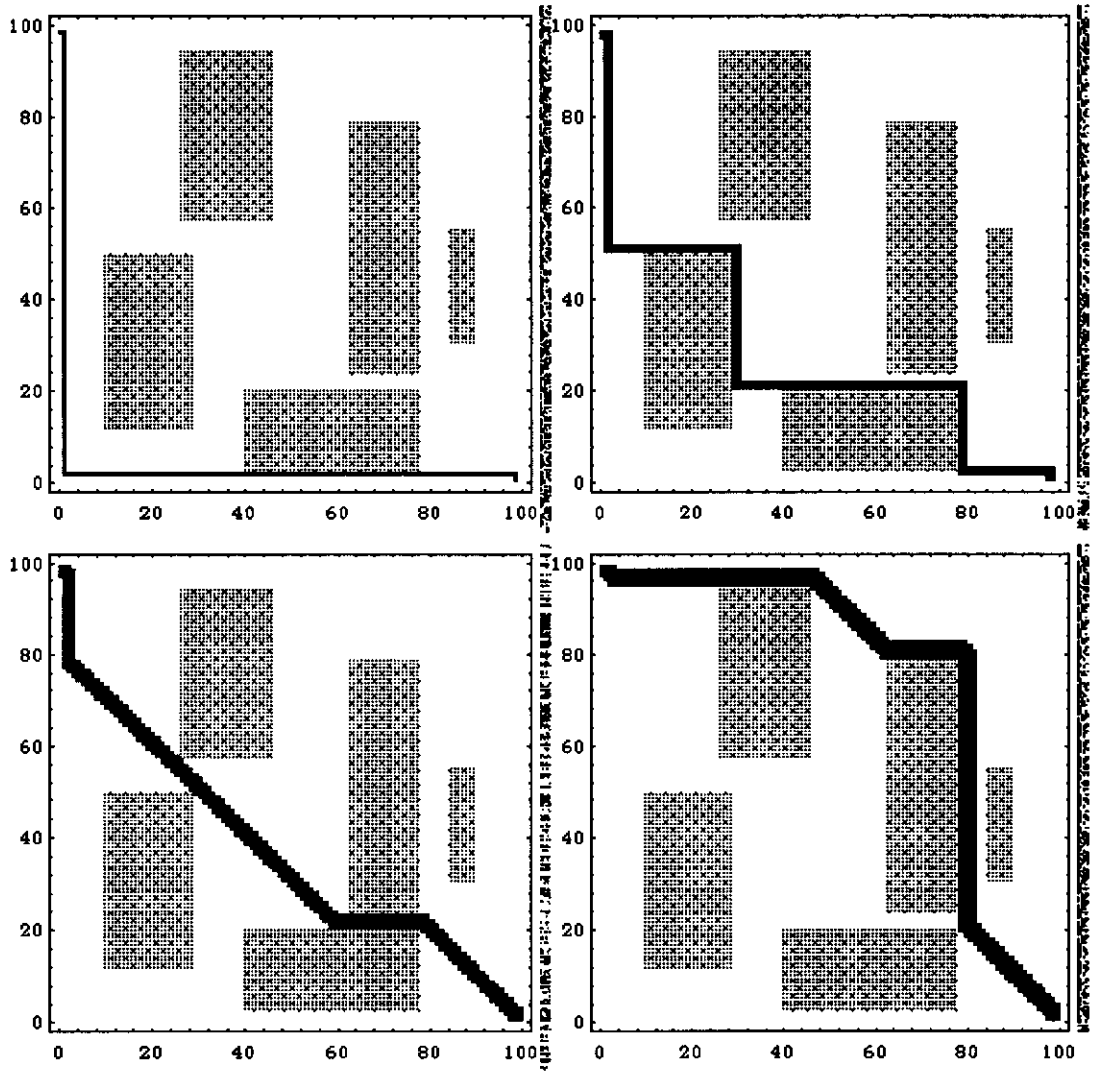

Figure 8: Robust motion planning in a region with polygonal obstacles. Note that the topology of the solutions changes as the width parameter is increased. The solutions shown correspond to widths $d = 1$ (top left), $d = 2$ (top right), $d = 3$ (bottom left), and $d = 4$ (bottom right).

Finally, we tested our methodology on random terrains, using a mesh resolution of 100 by 100 nodes and a range of d values. Each random terrain instance was generated as follows. All nodes in the mesh were initially assigned a weight of zero, except for a small random subset of the nodes which were each given large random positive weights. Then, a weight redistribution

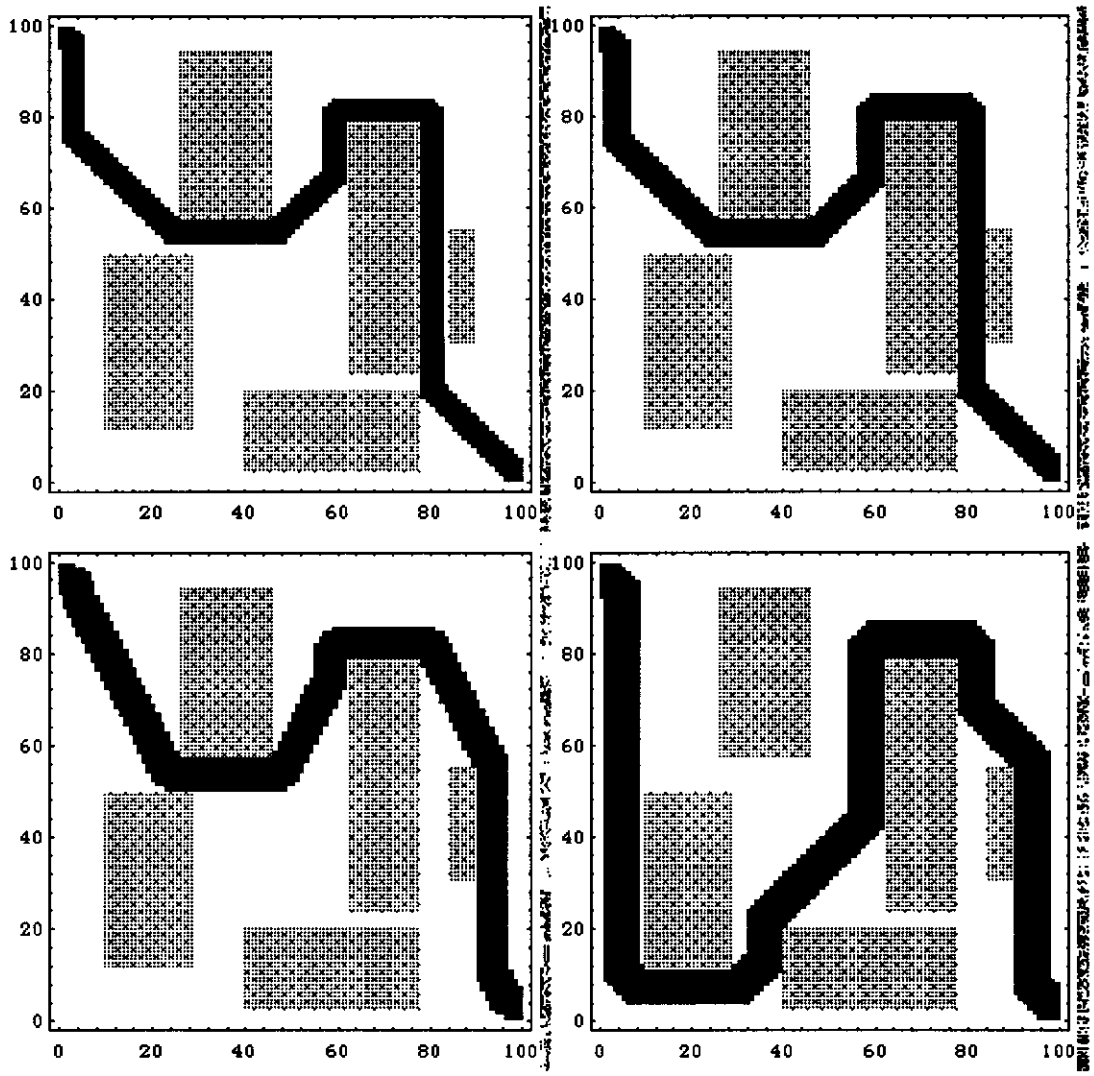

Figure 9: Robust motion planning in a region with polygonal obstacles (continued). The solutions shown correspond to widths $d = 5$ (top left), $d = 6$ (top right), $d = 7$ (bottom left), and $d = 8$ (bottom right).

step was iteratively used to increment each node's weight by a small random fraction of the total weight of its immediate neighbors until a smooth random terrain was obtained. Figures 10 and 11 depict typical results of the application of our algorithm to the RMPPG problem in a random terrain. Regions of greater weight are denoted by darker hues, and regions of smaller weight are depicted by lighter densities. The optimum width-d path is highlighted

in black. We emphasize that even with the Dinic algorithm, which is certainly not the ideal implementation for a mesh network topology, typical running times used to generate and solve all of the above classes of instances are on the order of only a few minutes on a SUN Sparc IPC. We therefore conclude that our approach constitutes a viable new method for optimal robust motion planning problem in general terrains.

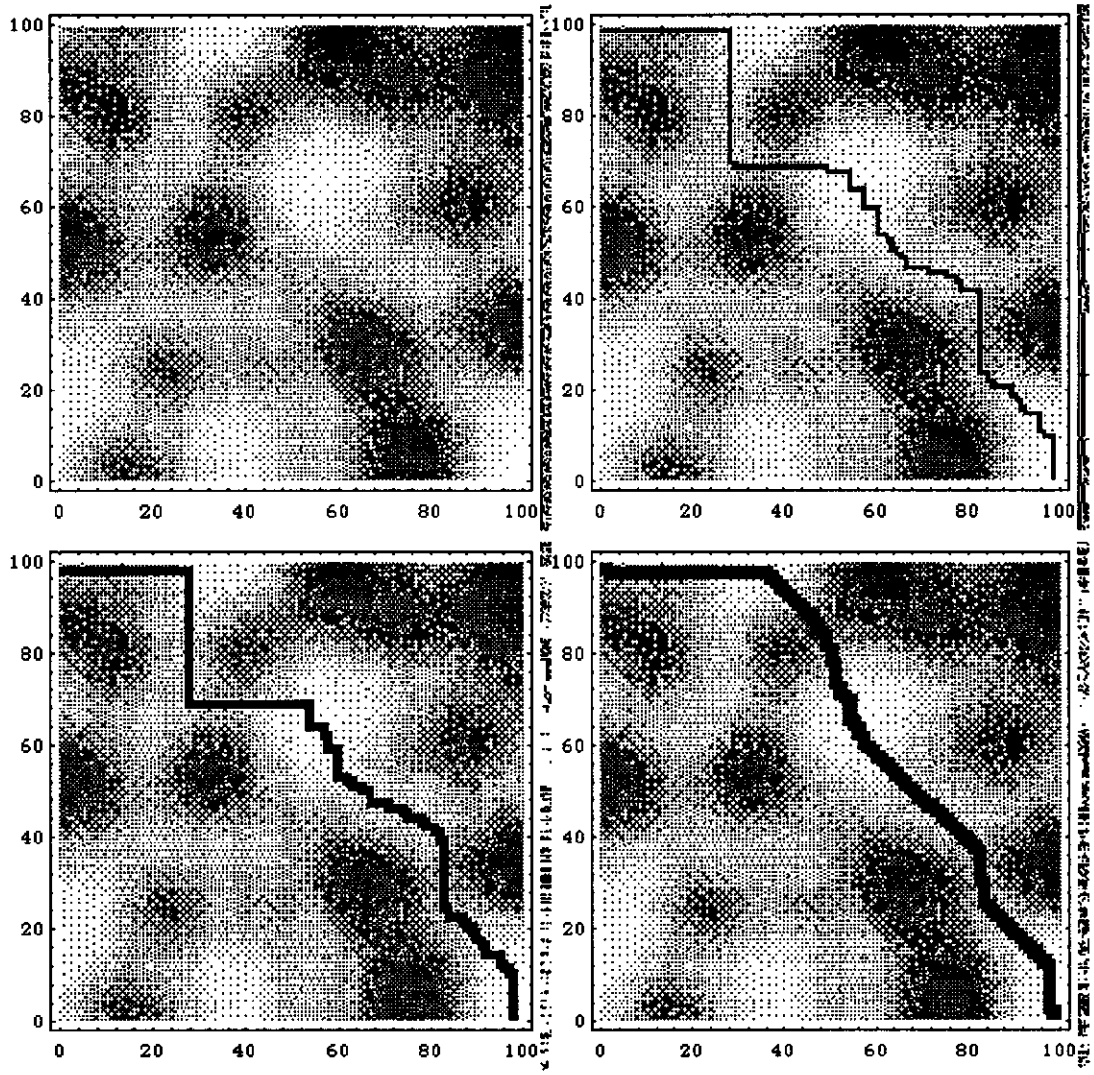

Figure 10: A randomly generated smooth terrain and its robust motion planning solutions. We see the terrain itself (top left), as well as solutions corresponding to widths $d = 1$ (top right), $d = 2$ (bottom left), and $d = 3$ (bottom right).

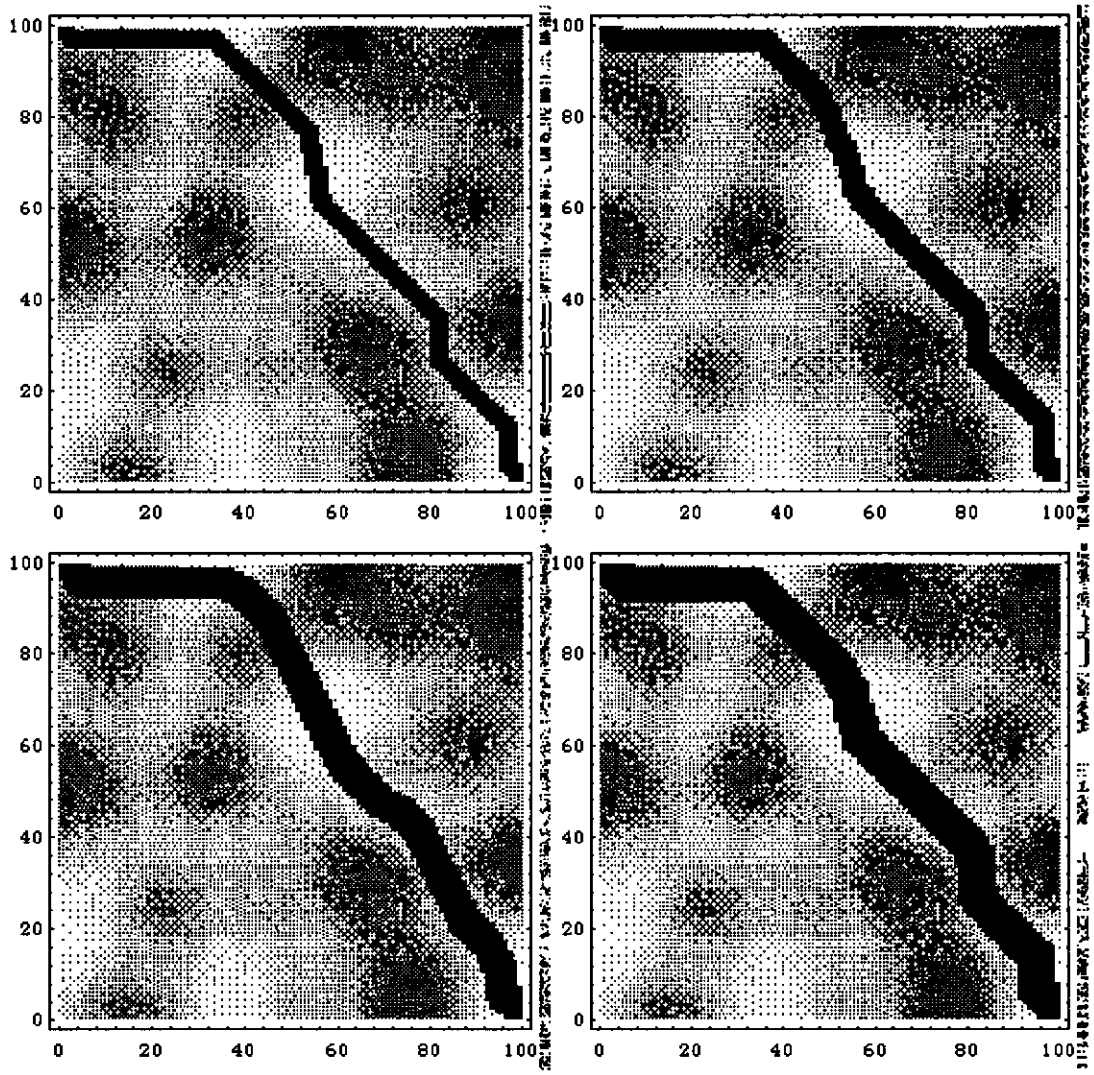

Figure 11: A randomly generated smooth terrain and its optimal robust motion planning solutions (continued). Here we see the solutions corresponding to widths $d = 4$ (top left), $d = 5$ (top right), $d = 7$ (bottom left), and $d = 8$ (bottom right).

4 Conclusion and Open Problems

We have developed a polynomial time algorithm which gives optimal solutions to the robust motion planning problem in a discretized grid environment. Our method is based on the duality between connecting paths and separating sets, and relies on a maximum-flow computation to

find a minimum-cost path of prescribed width in an arbitrarily weighted region. The accuracy of the solution with respect to the continuous version of the problem depends on the gridsize, which is intrinsic to the input. Our method enjoys a number of generalizations and additional applications, including a generalization to the discrete Plateau problem [13].

Chief among the future research directions is to improve the time complexity of the network flow routine; substantial improvement is likely to be achievable here since the mesh is a highly regular and symmetric graph, and it may be specified parametrically in a very concise form. Additional research might also entail robust path planning for simultaneous multiple agents, as well as addressing more general robot path planning issues, such as (i) incorporation of kinematic considerations, (ii) use of hierarchical approaches as a heuristic speedup, and (iii) addressing the case where the endpoints of the path are not on the boundary of the region.

References

[1] R. K. AHUJA, T. L. MAGNANTI, AND J. B. ORLIN, *Some recent advances in network flows*, SIAM Review, 33 (1991), pp. 175–219.

[2] R. K. AHUJA, J. B. ORLIN, AND R. E. TARJAN, *Improved time bounds for the maximum flow problem*, SIAM J. Computing, 18 (1989), pp. 939–954.

[3] BARRAQUAND AND J. C. LATOMBE, *Robot motion planning: A distributed representation approach*, Tech. Rep. Technical Report STAN-CS-89-1257, Department of Computer Science, Stanford University, 1989.

[4] J. CANNY, *The Complexity of Robot Motion Planning*, MIT Press, 1988.

[5] D. W. CHO, *Certainty grid representation for robot navigation by a bayesian method*, Robotica, 8 (1990), pp. 159–165.

[6] J. H. CONNELL, *Minimalist Mobile Robotics*, Academic Press, 1990.

[7] T. H. CORMEN, C. E. LEISERSON, AND R. RIVEST, *Introduction to Algorithms*, MIT Press, 1990.

[8] T. DEAN AND M. BODDY, *An analysis of time-dependent planning*, in Proc. AAAI, St. Paul, 1988, pp. 49–54.

[9] B. R. DONALD, *Error Detection and Recovery for Robot Motion Planning with Uncertainty*, Ph.D. dissertation, Dept. of EE&CS, MIT, Dept. of EE&CS, 1987.

[10] L. R. FORD AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, 1961.

[11] A. V. GOLDBERG AND R. E. TARJAN, *A new approach to the maximum flow problem*, in Proc. 18th ACM Symp. on Theory of Computing, 1986, pp. pp. 136–146.

[12] D. GOLDFARB AND M. D. GRIGORIADIS, *A computational comparison of the dinic and network simplex methods for maximum flow*, Annals of Operation Research, 13 (1988), pp. 83–123.

[13] R. E. GOMORY, T. C. HU, A. B. KAHNG, AND G. ROBINS, *Optimal solution of the discrete plateau problem.* manuscript, 1991.

[14] R. E. GOMORY, T. C. HU, AND J. M. YOHE, *r-separating sets*, Can. J. Math., XXVI (1974), pp. 1418–1429.

[15] T. C. HU, *Integer Programming and Network Flows*, Addison-Wesley, Reading, Mass., 1969.

[16] O. KHATIB, *Real-time obstacle avoidance for manipulators and mobile robots*, Intl. J. of Robotics Research, 5 (1986), pp. 90–98.

[17] J. C. LATOMBE, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[18] T. LOZANO-PEREZ, *Spatial planning: A configuration space approach*, IEEE Trans. on Computers, C-32 (1983), pp. 108–120.

[19] W. NELSON, *Continuous-curvature paths for autonomous vehicles*, in Proc. IEEE Intl. Conf. on Robotics and Automation, April 1989, pp. 1260–1264.

[20] C. O'DUNLAING AND C. K. YAP, *A retraction method for planning the motion of a disc*, J. Algorithms, 6 (1983), pp. 187–192.

[21] J. T. SCHWARTZ, M. SHARIR, AND H. HOPCROFT, *Planning, Geometry and Complexity of Robot Motion*, Ablex Publishing Corp., 1987.

[22] S. H. SUH AND K. G. SHIN, *A variational dynamic programming approach to robot-path planning with a distance-safety criterion*, IEEE Trans. on Robotics and Automation, 4 (1988), pp. 334–349.

[23] T. TSUCHIYA, *A note on discrete solutions of the plateau problem*, Mathematics of Computation, 54 (1990), pp. 131–138.

[24] S. WOLFRAM, *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley, Redwood City, CA, 1991. Second Edition.

[25] D. ZHU AND J. C. LATOMBE, *New heuristic algorithms for efficient hierarchical path planning*, IEEE Transactions on Robotics and Automation, 7 (1991), pp. 9–20.